Implementación de una técnica de aprendizaje máquina sin el uso de un framework

Oskar Adolfo Villa López

Abstract

En este trabajo se construye una red neuronal para un problema de segmentación de zonas inundadas. La red utiliza una arquitectura U-Net con transfer learning. Se realiza una comparativa de diferentes modelos utilizados como encoder: EfficientNet, ResNet, VGG y MobileNet.

Introducción

En este reporte se busca resolver un problema de segmentación de imágenes utilizando técnicas de deep learning. Las imágenes corresponden a fotos aéreas de zonas con inundaciones, y se busca predecir una máscara que indica los pixeles de la imagen en las que hay agua. El objetivo del modelo es ser utilizado para la planeación y toma de decisiones para dar respuesta a desastres naturales de este tipo.

Obtención y tratamiento de datos

Extracción y descripción

El conjunto de datos se obtuvo de la plataforma Kaggle [4]. Se cuenta con 290 imágenes, con sus respectivas máscaras. Las imágenes tienen dimensiones variables, por lo que es necesario escalarlas antes de entrenar el modelo y probarlo. Las imágenes están en formato RGB, y las máscaras son imágenes con valores en blanco para la segmentación y negro para el fondo.

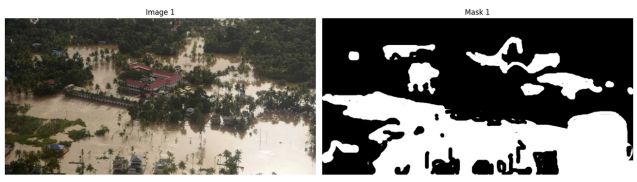

Figura 1: Ejemplo de imagen y máscara.

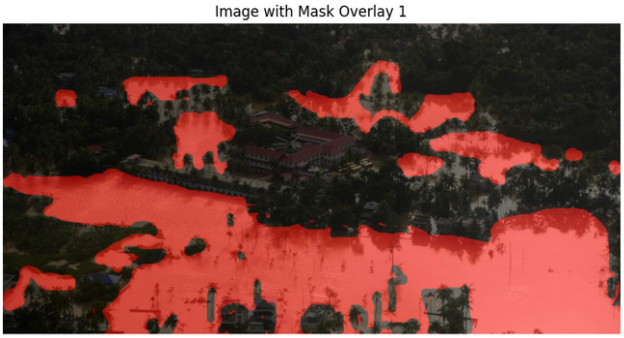

Figura 2: Visualización de máscara sobre imagen.

Escalamiento de datos

Se utilizó una función para escalar las fotos a un tamaño de 224 x 224 px. Esto es un paso necesario, ya que las imágenes deben tener todas el mismo tamaño para poder alimentar el modelo. El escalamiento se realizó mediante la librería Albumentations, que se utilizó para hacer transformaciones a las imágenes en el entrenamiento, lo cual se detalla más adelante. Para mantener el código más sencillo, se incluyó el escalamiento como parte de las transformaciones, y se aplicó también a los conjuntos de validación y pruebas.

Selección de conjuntos de entrenamiento y pruebas

Se dividió el conjunto de imágenes en 80 % entrenamiento y 20 % pruebas. Posteriormente, se dividió el conjunto de entrenamiento en 90 % entrenamiento y 10 % validación. Esto se decidió de esa manera por la poca cantidad de imágenes con las que se cuenta. Un conjunto de validación más grande sería dañino al eliminar imágenes de entrenamiento. Se utilizó Scikit-learn para la separación aleatoria de las imágenes en los conjuntos.

Se seleccionaron conjuntos de los siguientes tamaños:

- **Entrenamiento**: 208 imágenes. 72 %
- **Validación**: 24 imágenes. 8 %
- **Pruebas**: 58 imágenes. 20 %

Transformaciones

A través de la librería `Albumentations`, se aplicaron diversas transformaciones a las imágenes para mejorar

la capacidad de generalización del modelo. Estas transformaciones se aplican tanto a la imagen como a la máscara, manteniendo así la coherencia espacial entre ambas. Las transformaciones empleadas en el código son `Rotate`, `RandomSizedCrop` y `HorizontalFlip`, cada una configurada con parámetros específicos para ajustar el tamaño y orientación de las imágenes de entrada. A continuación, se detallan las características y parámetros de cada una:

- **Rotate**: Esta transformación rota la imagen de manera aleatoria hasta un límite de 360 grados, lo que permite que el modelo aprenda a reconocer el objeto en diferentes orientaciones. La probabilidad de aplicar esta rotación es del 90 %, y el modo de borde utilizado es `BORDER_REPLICATE`, que extiende el borde de la imagen para evitar vacíos.
- **RandomSizedCrop**: Con esta transformación, se realizan recortes aleatorios de la imagen, definiendo un tamaño mínimo y máximo para la altura del recorte. Este recorte luego se escala al tamaño de salida requerido, en este caso 224x224 píxeles. Los parámetros utilizados para esta transformación son:
  - **Altura mínima**: 224 $\times$ 75 % (o el 75 % de la altura original).
  - **Altura máxima**: 224 (la altura máxima de la imagen original).
  - **Tamaño de salida**: 224x224 píxeles, asegurando uniformidad en el tamaño final de las imágenes después del recorte.
- **HorizontalFlip**: Esta transformación refleja la imagen de forma horizontal con una probabilidad del 50 %, lo cual ayuda al modelo a aprender a reconocer los patrones en ambas direcciones y aumenta la variabilidad de los datos.

Metodología

Para la solución del problema, se utiliza un modelo de U-Net con transfer learning. Se realiza una comparativa entre 4 distintos encoders. Todos los modelos entrenan con los mismos hiperparámetros. Los encoders utilizados son los mismos utilizados en el paper *Efficient U-Net Architecture with Multiple Encoders and Attention Mechanism Decoders for Brain Tumor Segmentation* [2]:

- ResNet.
- VGG.
- MobileNet V2

En el paper citado se utilizan los 3 encoders de manera secuencial para un solo modelo. En este caso se busca analizar sus resultados de manera independiente. Además, se incluye EfficientNet, ya que se encontró su uso en múltiples papers con buenos resultados.

Para la comparación, se generaron gráficas del comportamiento del modelo durante el entrenamiento para error, accuracy y Mean Intersection Over Union (mIoU). Para comparar los resultados con el conjunto de pruebas, se utiliza accuracy y mIoU.

Arquitectura

Modelo

El modelo utilizado es U-Net. Este modelo es ampliamente utilizado en problemas de clasificación. Consiste en una arquitectura de redes neuronales convolucionales, que contiene 2 secciones:

- Encoder: El encoder es la primera sección de la arquitectura. Se encarga de disminuir las dimensiones de la imagen, extrayendo características distintivas en cada capa.
- Decoder: Tiene una estructura inversa al encoder. Se encarga de construir la máscara a partir de la deconstrucción del encoder.

Ambas secciones están conectadas de manera secuencial, pero cuentan también con conexiones intermedias llamadas *skip connections*, las cuales conectan cada capa del encoder con su capa de decoder correspondiente. Las *skip connections* ayudan a mantener detalles de dimensionalidad alta en el proceso de deconstrucción de la imagen [2].

La arquitectura en código viene de la librería Segmentation Models, que es una extensión de PyTorch. El modelo incluye la opción de utilizar optimizadores, *schedulers* para *learning rate* y *weight decay* como método de regularización.

Hiperparámetros

Los hiperparámetros a utilizar son los siguientes:

- Épocas: 300. En el entrenamiento se observó que los modelos alcanzan resultados estables a partir de las 80 épocas. Se subió el número a 100 para buscar mejores resultados.
- Max learning rate: 0.01. Es un valor comunmente usado en este tipo de modelos. Además, se realizaron pequeñas pruebas antes del entrenamiento, en donde este valor dio mejores resultados.
- Función de pérdida: Cross Entropy Loss.
- Optimizador: Adam.
- Learning Rate Scheduler: One Cycle Policy.

Para el entrenamiento se utilizaron batches de 32 imagenes, por medio de un Dataloader de PyTorch.

Primera iteración: EfficentNet

Encoder

EfficientNet es una arquitectura que utiliza un escalamiento uniforme de dimensiones de profundidad, ancho y resolución. Se basa en la intuición de que mientras más grande sea una imagen, se necesitan más capas para extraer sus características [1]. Para el entrenamiento se utilizó el modelo *efficientnet-b2*, que cuenta con 7 millones de parámetros. Los pesos utilizados son los del entrenamiento con ImageNet [3].

Entrenamiento

El comportamiento del entrenamiento se muestra en las Figuras 3, 4 y 5. El modelo tuvo dificultades a lo largo del entrenamiento para hacer predicciones en el conjunto de validación, lo cual es observable en los 3 indicadores. Al final del entrenamiento se llegó a valores estables, con una ligera ventaja para el conjunto de entrenamiento.

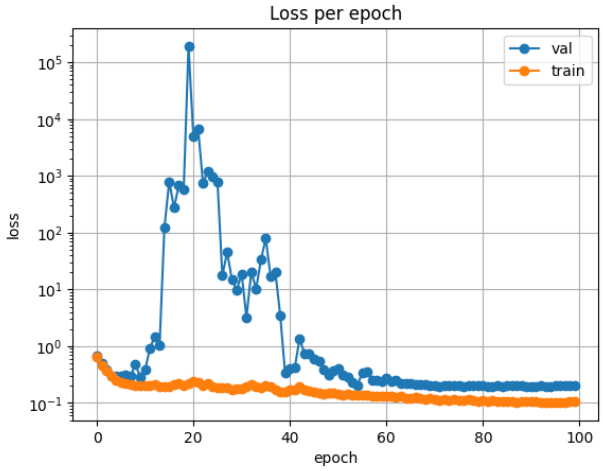

Figura 3: Error a lo largo del entrenamiento del modelo con EfficientNet.

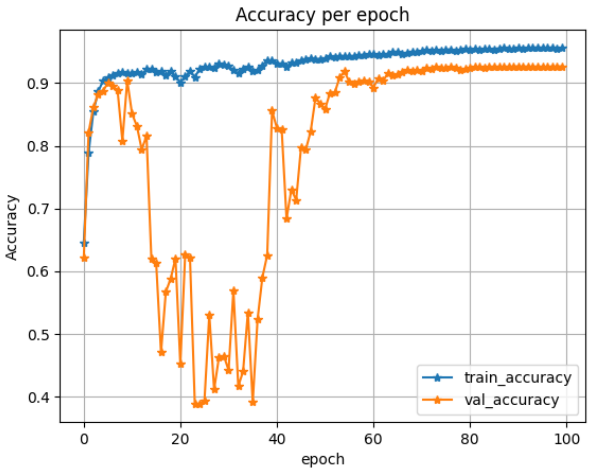

Figura 4: Accuracy a lo largo del entrenamiento del modelo con EfficientNet.

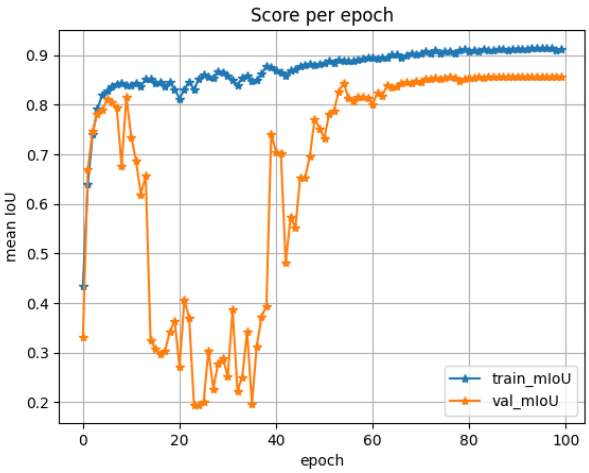

Figura 5: mIoU a lo largo del entrenamiento del modelo con EfficientNet.

Evaluación del modelo

Se evaluó el modelo con el conjunto de pruebas, y se obtuvieron los siguientes resultados:

- **mIoU**: 0.8131
- **Accuracy**: 0.9137

Debido a que éste es un problema de segmentación, mIoU es una medida más adecuada para evaluar el modelo. El valor del conjunto de pruebas es un poco más pequeño al de validación en la última época de entrenamiento (alrededor de 0.85). Sin embargo, se debe considerar que el conjunto de validación es muy pequeño, por lo que puede que exista un sesgo. El modelo puede considerarse como *fitted* debido a que muestra resultados muy similares tanto en entrenamiento como en pruebas, y otorga resultados con una alta calificación. En la Figura 6 se muestran predicciones realizadas por el modelo.

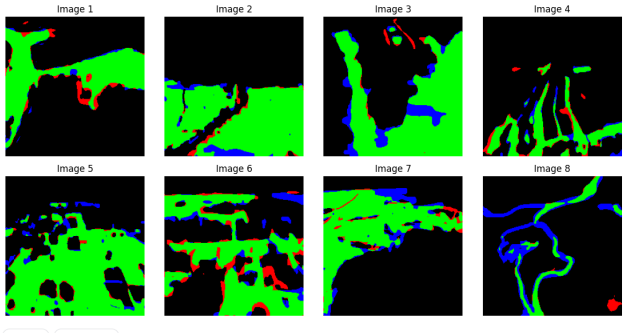

Figura 6: Predicciones del modelo con EfficientNet. (Verde: Predicción correcta, Rojo: Falso positivo, Azul: Falso negativo

Segunda iteración: ResNet

Encoder

El modelo ResNet obtiene su nombre del término red residual. Los residuos son conexiones directas entre capas no contiguas. Esto permite hacer redes más profundas, ya que combate el problema de los *vanishing gradients*, que dificultan el aprendizaje en el entrenamiento [2]. El modelo utilizado es *resnet50*, que tiene 23 millones de parámetros. Se utilizaron los pesos del entrenamiento con ImageNet [3].

Entrenamiento

El comportamiento del entrenamiento se muestra en las Figuras 7, 8 y 9. Los 3 indicadores tienen valores muy variados en las primeras 40 épocas, los cuales se estabilizan posteriormente. Los indicadores de entrenamiento y validación son muy similares, lo que indica que el modelo no está realizando un *overfitting*.

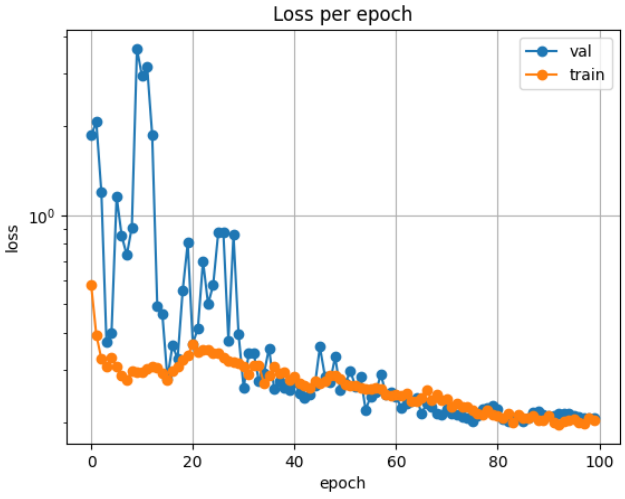

Figura 7: Error a lo largo del entrenamiento del modelo con ResNet 50.

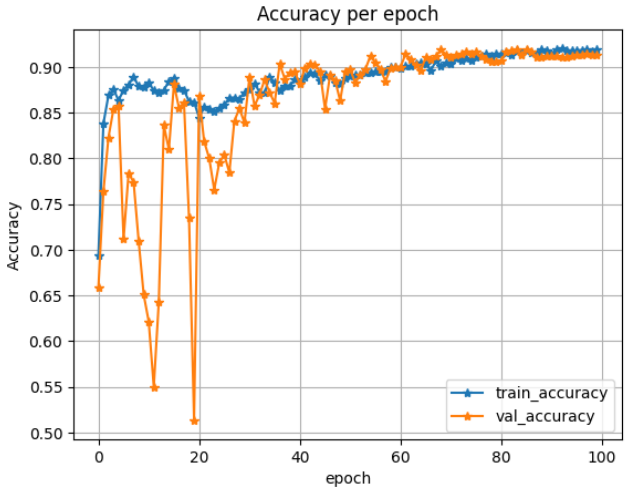

Figura 8: Accuracy a lo largo del entrenamiento del modelo con ResNet 50.

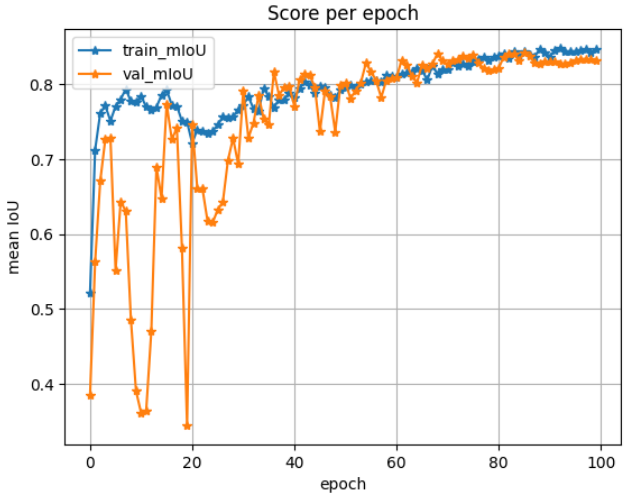

Figura 9: mIoU a lo largo del entrenamiento del modelo con ResNet 50.

Evaluación del modelo

Se evaluó el modelo con el conjunto de pruebas, y se obtuvieron los siguientes resultados:

- **mIoU**: 0.7728
- **Accuracy**: 0.8921

Al igual que la iteración anterior, el resultado es ligeramente inferior con el conjunto de pruebas que con entrenamiento y validación. Sin embargo, se encuentra dentro de un rango aceptable, y se puede considerar el modelo como *fit*. En la Figura 10 se muestran predicciones del modelo.

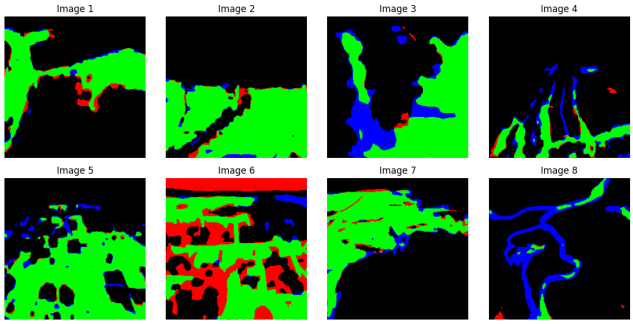

Figura 10: Predicciones del modelo con ResNet 50. (Verde: Predicción correcta, Rojo: Falso positivo, Azul: Falso negativo

Tercera iteración: VGG

Encoder

El modelo VGG se compone de pequeñas capas convolucionales de 3x3 [2]. El modelo utilizado es *vgg19*, que se compone de 19 capas y contiene 20 millones de parámetros, con los pesos del entrenamiento con ImageNet [3].

Entrenamiento

El comportamiento del entrenamiento se muestra en las Figuras 11, 12 y 13. El entrenamiento es más estable comparado con los modelos anteriores. Al final, se alcanzan valores muy similares para los conjuntos de entrenamiento y validación.

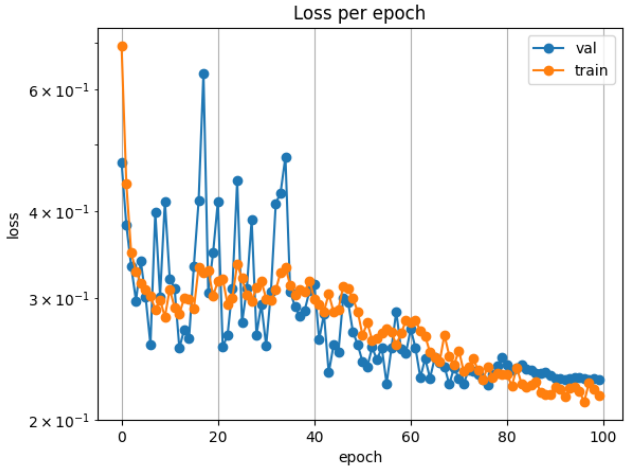

Figura 11: Error a lo largo del entrenamiento del modelo con VGG19.

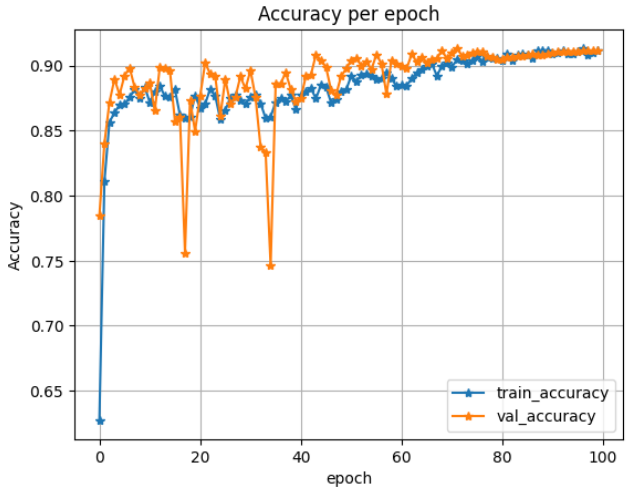

Figura 12: Accuracy a lo largo del entrenamiento del modelo con VGG19.

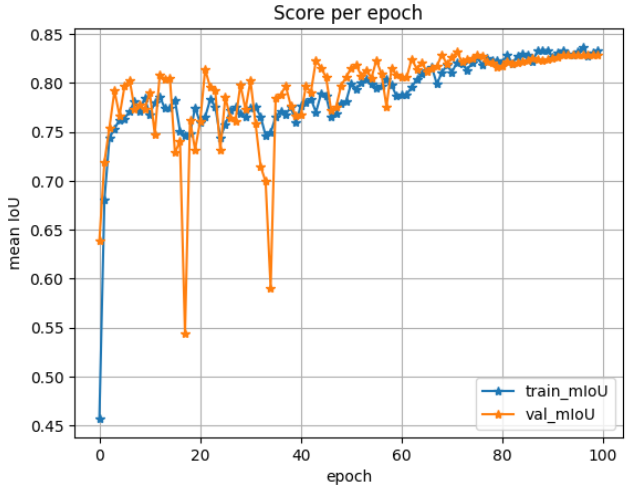

Figura 13: mIoU a lo largo del entrenamiento del modelo con VGG19.

Evaluación del modelo

Se evaluó el modelo con el conjunto de pruebas, y se obtuvieron los siguientes resultados:

- **mIoU**: 0.7561
- **Accuracy**: 0.8824

El valor de validación difiere del valor de entrenamiento, aunque en esta ocasión la diferencia es un poco mayor, sobre todo en mIoU. El modelo se puede clasificar como un ligero *overfit*. En la Figura 11 se muestran predicciones del modelo.

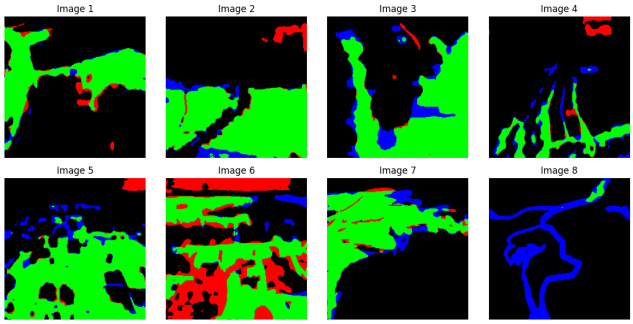

Figura 14: Predicciones del modelo con VGG19. (Verde: Predicción correcta, Rojo: Falso positivo, Azul: Falso negativo

Cuarta iteración: MobileNet

Encoder

MobileNetV2 es una versión mejorada de la red MobileNetV1, diseñada para optimizar la eficiencia en términos de complejidad y tamaño del modelo.

Mientras que MobileNetV1 se basa en *convoluciones separables en profundidad* en la primera capa para reducir el costo computacional, y usa convoluciones 1×1 en la segunda capa para combinar los canales de entrada y crear nuevas características, MobileNetV2 utiliza dos tipos de bloques para mejorar el rendimiento:

- **Bloque residual con *stride* 1**: permite conservar la resolución espacial de la entrada.
- **Bloque con *stride* 2**: permite reducir la resolución espacial, útil para reducir el tamaño de los mapas de características a medida que avanza el flujo de datos en la red.

Cada bloque en MobileNetV2 consta de tres capas:

1. Capa de **convolución** 1×1 **con una función de activación ReLU6**, que ayuda a mantener la estabilidad numérica.
2. Capa de **convolución *depthwise*** (que opera de forma independiente en cada canal).
3. Capa de **convolución** 1×1 sin activación, para la combinación lineal de los canales. [2]

El modelo utilizado es *mobilenet-v2*, con 2 millones de parámetros entrenados con ImageNet [3].

Entrenamiento

En las Figuras 15, 16 y 17 se observa el comportamiento durante el entrenamiento. El modelo se comporta de manera estable, con saltos en las épocas 20 a 40.

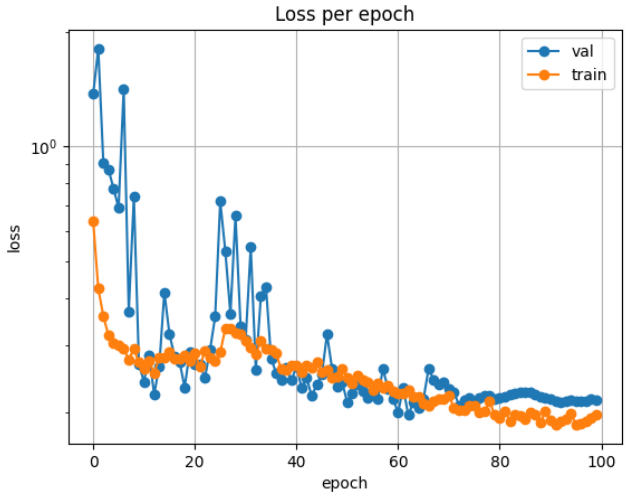

Figura 15: Error a lo largo del entrenamiento del modelo con MobileNet V2.

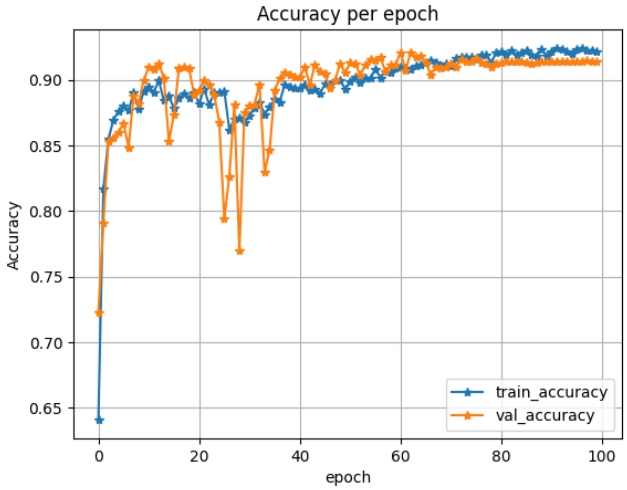

Figura 16: Accuracy a lo largo del entrenamiento del modelo con MobileNet V2.

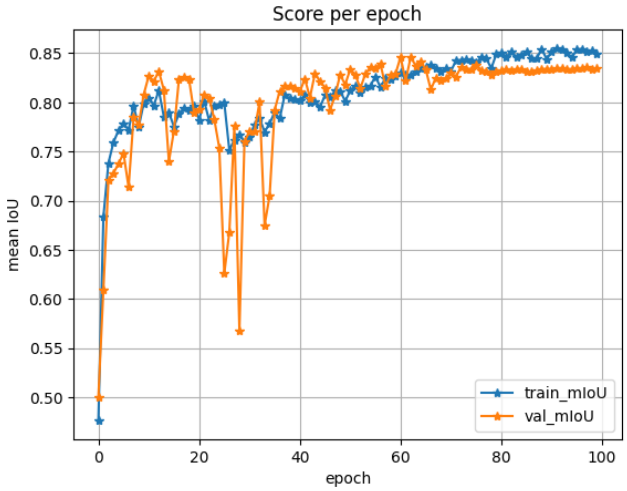

Figura 17: mIoU a lo largo del entrenamiento del modelo con MobileNet V2.

Evaluación del modelo

Se evaluó el modelo con el conjunto de pruebas, y se obtuvieron los siguientes resultados:

- **mIoU**: 0.7817
- **Accuracy**: 0.8948

El modelo muestra una ligera diferencia entre los valores de validación y pruebas, pero está dentro de un rango aceptable. El modelo se considera como *fitted*. En la Figura 18 se observan predicciones del modelo.

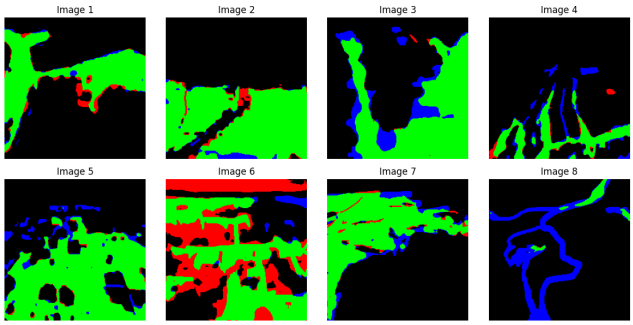

Figura 18: Predicciones del modelo con MobileNet V2. (Verde: Predicción correcta, Rojo: Falso positivo, Azul: Falso negativo

Resultados generales

En la Tabla 1 se muestra la comparación de las métricas de los cuatro modelos. El modelo que utiliza EfficientNet como encoder tiene mejores resultados tanto en mIoU como en accuracy.

Tabla 1: Accuracy y mIoU de tres modelos diferentes

Modelo	mIoU	Accuracy
EfficientNet	0.8131	0.9137
ResNet	0.7728	0.8921
VGG	0.7561	0.8824
MobileNet	0.7817	0.8948

Conclusiones

Los cuatro modelos logran realizar predicciones aceptables. EfficientNet tiene una clara ventaja, mientras que el resto de los modelos tienen resultados muy similares. Además, el número de parámetros de EfficientNet también se encuentra entre los más ligeros, solo por detrás de MobileNet, por lo que es un buen balance entre complejidad y eficiencia.

Referencias

[1] Mingxing Tan y Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". En: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019. DOI: 10.48550/arXiv.1905.11946. URL: https://paperswithcode.com/paper/efficientnet-rethinking-model-scaling-for (visitado 31-10-2024).

[2] John M. Valanarasu, Vishwanath A. Sindagi y Vishal M. Patel. "Efficient U-Net Architecture with Multiple Encoders and Attention Mechanism Decoders for Brain Tumor Segmentation". En: *Diagnostics* 13.5 (2023), pág. 872. DOI: 10.3390/diagnostics13050872. URL: https://www.mdpi.com/2075-4418/13/5/872 (visitado 31-10-2024).

[3] SMP Documentation. *Encoders — Segmentation Models*. 2024. URL: https://smp.readthedocs.io/en/latest/encoders.html (visitado 31-10-2024).

[4] Niyar R Barman Faizal Karim Krish Sharma. *Flood Area Segmentation*. URL: https://www.kaggle.com/datasets/faizalkarim/flood-area-segmentation (visitado 09-10-2024).